Licenciatura Engenharia Informática e Multimédia

Infraestruturas Computacionais Distribuídas

Semestre de Verão 2022 / 2023

Trabalho Prático 1

Docente Diogo Remédios

08 de Maio de 2023

Trabalho realizado por:

Fábio Dias, nº 42921

Samuel Ribeiro, nº 49249

Índice

Índice de Figuras

1. Introdução

Para a primeira fase do trabalho foi-nos pedido para desenvolver o jogo "Quatro-em-Linha", sendo este realizado entre dois jogadores diferentes e em máquinas diferentes. O jogo deve possuir oito linhas e oito colunas.

Para isto, devemos desenvolver duas estruturas: Cliente e Servidor. O Servidor possuirá a informação de todos os Clientes previamente registados, assim como a informação dos Clientes activos no momento. Ainda possuirá a responsabilidade de iniciar os jogos e guardar as informações sobre todos os jogos a decorrerem. O Cliente possuirá a representação visual da lista dos clientes activos assim como o estado de jogo actual, caso se encontre num.

O Cliente possui um nome, password, nacionalidade, idade e uma fotografia, assim como o registo de vitórias, registo de derrotas e tempo gasto em cada jogo.

2. Redes

Dado que a intenção do trabalho é poder jogar em máquinas diferentes, é preciso possuir uma noção de como funcionam as Redes. Uma rede de computadores é um sistema de comunicação que liga as máquinas uma a outra, vulgarmente designadas por *hosts*[1].

Não existe a garantia que os tipos de dados enviados e recebidos possuam a mesma dimensão, logo, uma forma de solucionar este problema é possuir o mesmo *encoding*, isto é, associar os valores numéricos recebidos a diferentes caracteres. Uma forma eficiente e organizada de fazer isto, é transferir ficheiros XML, onde é definido o tipo de *encoding* que pretendemos para aquele ficheiro.

Apresentados estes conceitos e ideias, podemos concluir que a transferência de informação entre o Servidor e o Cliente foi feita a partir de documentos XML, que desenvolveremos mais à frente.

3. Arquitectura Cliente/Servidor

Como mencionado, serão precisas duas estruturas: Cliente e Servidor. Desta forma, é possível desenvolver a arquitectura Cliente/Servidor. Esta arquitectura é típica num clima de redes.

O Servidor é um processo que espera ser contactado pelos processos Cliente e tem como objectivo responder aos seus pedidos. Estes podem ser divididos em dois tipos: Iterativo e Concorrente.[1]

Um Servidor Iterativo é aquele que ao receber um pedido, trata-o directamente e unicamente. Isto é uma simples para quando o processamento do pedido demora pouco tempo e não existe concorrência de pedidos. Por sua vez, um Servidor Concorrente cria um novo processo ou tarefa para responder aos pedidos recebidos. Desta forma, nenhum pedido é deixado em espera. Este último tipo de Servidor é o desejado para o nosso trabalho.

Também existem dois tipos de Clientes: *Thin* e *Thick*.

O Cliente *Thin* é responsável por apresentar a informação ao jogador, seja a partir de texto ou de algo gráfico. Desta forma, toda a lógica está no Servidor. Por sua vez, o Cliente *Thick* possui toda a lógica na aplicação. No caso do trabalho, optamos por termos um Cliente *Thin*, dado que as máquinas podem não ter tantos recursos computacionais e existe a possibilidade de existirem Clientes que não são de confiança, por exemplo, Clientes que não estão a usar a estrutura feita por nós, mas sim uma desenvolvida pelos mesmos, possibilitando assim batotas. Com a lógica toda do lado do Servidor, esta situação não é possível.

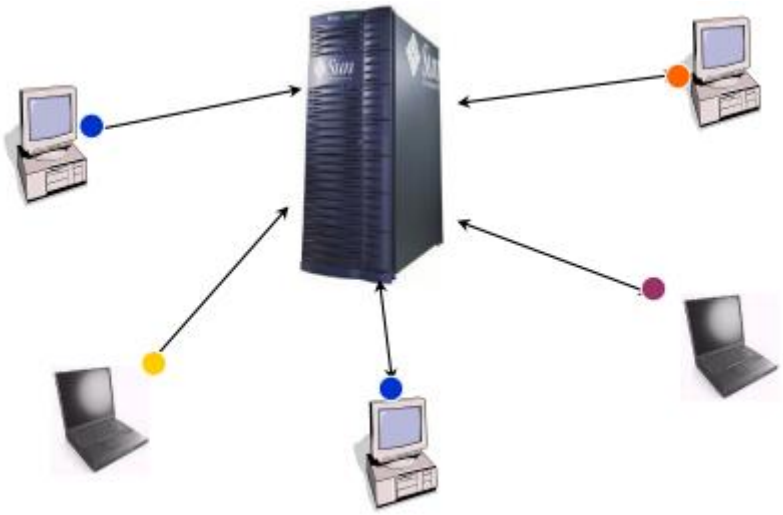

Figura 1 - Modelo Cliente/Servidor

4. Sockets

De forma a conseguirmos comunicar entre as máquinas, é necessário o uso de mecanismos de comunicação bidimensional. É aqui que entram os *Sockets*. Estes são caracterizados pelo protocolo que será usado, o endereço IP e o porto da máquina, assim como o endereço IP e o porto da máquina destino.[1]

O endereço IP é o identificador da máquina numa rede. No entanto, uma máquina pode ter mais do que um processo, cada um com o seu número de porto.

Dado isto, no trabalho, o Cliente vai abrir uma ligação, enviar e receber dados e, quando quiser, fechar a ligação. Por sua vez, o Servidor é associado a um porto, espera por dados e aceita ligações das máquinas.

5. Protocolos de Transporte

Como mencionado, um Socket também é caracterizado pelo seu protocolo. Os computadores ligados em rede usam protocolos para comunicarem. Um protocolo é um conjunto de regras e convenções às quais os intervenientes na comunicação devem obedecer, quando trocarem informações.

Existem diversos tipos de protocolos. No caso da comunicação em rede, é necessário dar foco ao nível de transporte, que é visível ao nível da aplicação. Neste nível existem dois tipos de serviços: Datagramas e Circuitos Virtuais.

O protocolo TCP (Transfer Control Protocol) é protocolo de Circuito Virtual. Este garante que tanto o Servidor como o Cliente recebem todos os dados pretendidos, preserva a ordem de transimissão dos mesmos e não admite pacotes duplicados. Caso estes pacotes de dados não sejam recebidos e, por sua vez, reconhecidos, os pacotes de dados são enviados novamente, evitando assim, a perda dos mesmos. Funciona de forma bidireccional e um processo pode ter múltiplas conexões.[1][2]

O protocolo UDP (User Datagram Protocol) é um protocolo de Datagramas. Este não possui uma ligação permanente. Também não é fiável dado que não existe a garantia da entrega de pactos, não preserva a ordem de transmissão e podem ser recebidos pacotes duplicados.[1][3]

Com estas informações, dado que pretendemos ter ligações permanentes e garantir a entrega de pacotes, decidimos usar o protocolo TCP.

6. Protocolo de Aplicação

De forma a garantir que a comunicação entre os Clientes e o Servidor fosse possível e válido, foi necessário criar um protocolo da aplicação, robusto e extenso, para precaver qualquer tipo de situação, tanto válida como inválida. Este recorre ao uso dos documentos XML e XSD.

O XML (Extensible Markup Language) é um documento baseado em texto com regras muito específicas. É possível definir o encoding usado para o documento, assim como organizar os dados de forma hierárquica, independentemente do software e hardware em questão. Neste documento existe a separação do conteúdo da formatação, a simplicidade na sua legibilidade e a possibilidade de criação de *tags* sem limitação[4][5].

Os documentos XSD (XML Schema Definition) é uma linguagem de base XML para especificar as estruturas de documentos XML, estas designadas por esquema. Um documento XML é definido como bem-formado se respeitar as regras da linguagem XML. Por sua vez, é definido como válido sempre que respeita todas as restrições do esquema e se é bem-formado.[6][7]

Recorrendo ao uso do XSD, podemos validar as mensagens recebidas por parte dos Clientes, dado que pode existir um Cliente simulado, ou seja, um Cliente não desenvolvido por nós que, pode ou não, desrespeitar as práticas expectáveis. Já o contrário, no sentido do Servidor para os Clientes, esta validação não será necessária pois temos a garantia de enviar mensagens bem-formadas por parte do Servidor, visto que foi implementado por nós. Como consequência, o Cliente fica ainda mais leve pois não necessita de validações constantes.

As mensagens que o Cliente deve enviar são: SignIn, que indica que o cliente pretende registar-se na base de dados do Servidor. Isto apenas é possível se o nome do Cliente ainda não existir na base de dados; Login, o Cliente pretende juntar-se à lista dos utilizadores e, consequentemente, receber os outros jogadores já ligados. Isto só se realiza caso o jogador já existir na base de dados e se a palavra-passe inserida for a associada ao nome do Cliente; ChallengePlayer, o Cliente desafia outro Cliente desde que seja possível; PlayColumn, escolhe a coluna em que o Cliente quer colocar a sua peça.

As mensagens que o Servidor deve enviar são: LoginAnswer, o Servidor devolve um parâmetro que notifica o utilizador caso o seu nome ou palavra-passe estejam incorrectos ou não existam. Caso tudo corra como devido, é devolvido a lista de Clientes activos; AddPlayer, É adicionado à actual lista de Clientes um novo Cliente que entretanto entrou na aplicação; ChallengePlayerAnswer, o Servidor envia ao Cliente que iniciou o desafio, a resposta do oponente. Caso tenha aceitado o desafio, irão ser movidos para o ecrã de jogo onde passaram a jogar um contra o outro; UpdateStatus, este atualiza o estado em que um específico Cliente se encontra. Seja este *Online*, *Challenged* ou até *InGame*; UpdateBoard, o Servidor envia aos clientes o estado em que se encontra o tabuleiro do jogo, assim como quem é a jogar no turno actual e ainda que botões devem ser activados.

```
<?xml version="1.0" encoding="UTF-8"?>
<Protocolo>
        <!-- Client -> Server -->
    <SignIn password="" playerName=""/>
    <Login password="" playerName=""/>
    <ChallengePlayer playerName=""/>
    <PlayColumn col="" player=""/>
        <!-- Server -> Client -->
  - <LoginAnswer validate="">
        <Player playerName="" jogosPerdidos="" jogosGanhos="" jogosJogados=""/>
    </LoginAnswer>
    <AddPlayer playerName="" jogosPerdidos="" jogosGanhos="" jogosJogados=""/>
    <ChallengePlayerAnswer playerName="" answer=""/>
    <UpdateStatus status="" playerName2="" playerName1=""/>
  + <UpdateBoard nextPlayerTurn="">
</Protocolo>
```

Figura 2 - Protocolo de Aplicação

7. Base de Dados

Para guardarmos as informações desejadas dos Clientes, foi necessário estabelecermos e preparamos uma base de dados. Esta será desenvolvida em XML, de forma a mantermos coerência entre o protocolo e a base de dados, em termos de linguagem.

Por cada novo Cliente que se regista na base de dados, é criado um novo elemento com o nome "Player". Este possui os atributos "nome", o nome com que o Cliente se registou; "password", a sua palavra-passe; "jogosJogados", o número de jogos já realizados por este Cliente, iniciados a zero; "jogosGanhos" e "jogosPerdidos", o número de jogos ganhos e perdidos, respectivamente, efectuados por este Cliente, ambos iniciados a zero. Como filhos deste elemento, tem-se a "Nacionalidade", a "Idade" e a "Fotografia" do Cliente.

Sempre que um Cliente se regista, é procurado o seu nome na base de dados e, caso não exista, o processo referido é efectado. Caso exista, é devolvida uma mensagem ao Cliente a informar do sucedido. No "Login", para além de verificarmos o nome, caso este venha positivo, confirmarmos se a sua palavra-passe corresponde à submetida na altura do registo.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayerDatabase>
  - <Player playerName="a" password="a" jogosPerdidos="0" jogosJogados="0" jogosGanhos="0">
        <Nacionalidade/>
        <Idade/>
        <Fotografia/>
    </Player>
  - <Player playerName="b" password="b" jogosPerdidos="0" jogosJogados="0" jogosGanhos="0">
        <Nacionalidade/>
        <Idade/>
        <Fotografia/>
    </Player>
  - <Player playerName="c" password="c" jogosPerdidos="0" jogosJogados="0" jogosGanhos="0">
        <Nacionalidade/>
        <Idade/>
        <Fotografia/>
    </Player>
  - <Player playerName="d" password="d" jogosPerdidos="0" jogosJogados="0" jogosGanhos="0">
        <Nacionalidade/>
        <Idade/>
        <Fotografia/>
    </Player>
</PlayerDatabase>
```

Figura 3 - Base de Dados

8. Desenvolvimento da Aplicação

De acordo com tudo o que já foi mencionado, podemos concluir que vamos precisar de duas estruturas cruciais: O Servidor e o Cliente.

O Servidor tem de ser Concorrente de forma a atender vários pedidos simultaneamente sem precisar de meter em espera pedidos dos Clientes. Ou seja, sempre que existir uma tentativa de ligação, é criada uma *thread* para atender os pedidos da mesma.

```
newSock = serverSocket.accept();

Thread th = new ThreadServidor(newSock, dados);
th.start();
```

Figura 4 – Criação de Thread Servidor

Por sua vez, os Clientes também vão ter de possuir um *thread* para receber mensagens oriundas do Servidor e a uma classe base para proceder ao envio de mensagens para o Servidor.

```
ThreadCliente thread = new ThreadCliente(this, socketReader);
thread.start();
```

Figura 5 - Criação de Thread Cliente

Também estabelecemos o nosso protocolo de comunicação e as diversas mensagens que tornam possível esta comunicação bidireccional. Para o efeito, criámos uma Classe de utilidade que possui todos os métodos apropriados para a comunicação.

```
//TODO Métodos Cliente -> Servidor
public static String SignIn(String playerName, String password)

public static String Login(String playerName, String password)

public static String ChallengePlayer(String playerName)

public static String PlayColumn(String player, int column)

//TODO Métodos Servidor -> Cliente
public static String LoginAnswer(boolean validate, int numberOfPlayers, String[] playersInformation)

public static String AddPlayer(String playerName, int jogosJogados, int jogosGanhos, int jogosPerdidos)

public static String ChallengePlayerAnswer(String playerName, boolean answer)

public static String UpdatedBoard(String[] board, String[] buttonColumns, String currentTurnPlayerName)

public static String updateStatus(String player1, String player2, String status)
```

Figura 6 - Classe Mensagem

As mensagens, enviadas por parte dos Clientes, têm de ser validadas com recurso ao XSD.

```
if(!Mensagem.ValidateXMLMessage(message))
{
    System.out.println("Mensagem Inválida");
    return;
}
```

Figura 7 - Validação com XSD

Existem mais três estruturas importantes para o correcto funcionamento do planeado: uma lista de Clientes, uma lista dos jogos existentes e a base de dados. Todas as estruturas podem estar encapsuladas numa só classe.

```
public class Dados {

    private Document playerDatabase;
    public ArrayList<RegistoCliente> listaClientes;
    private ArrayList<Jogo> jogos;
```

Figura 8 - Classe Dados

Após esta conclusão, podemos apresentar esta arquitectura num diagrama.

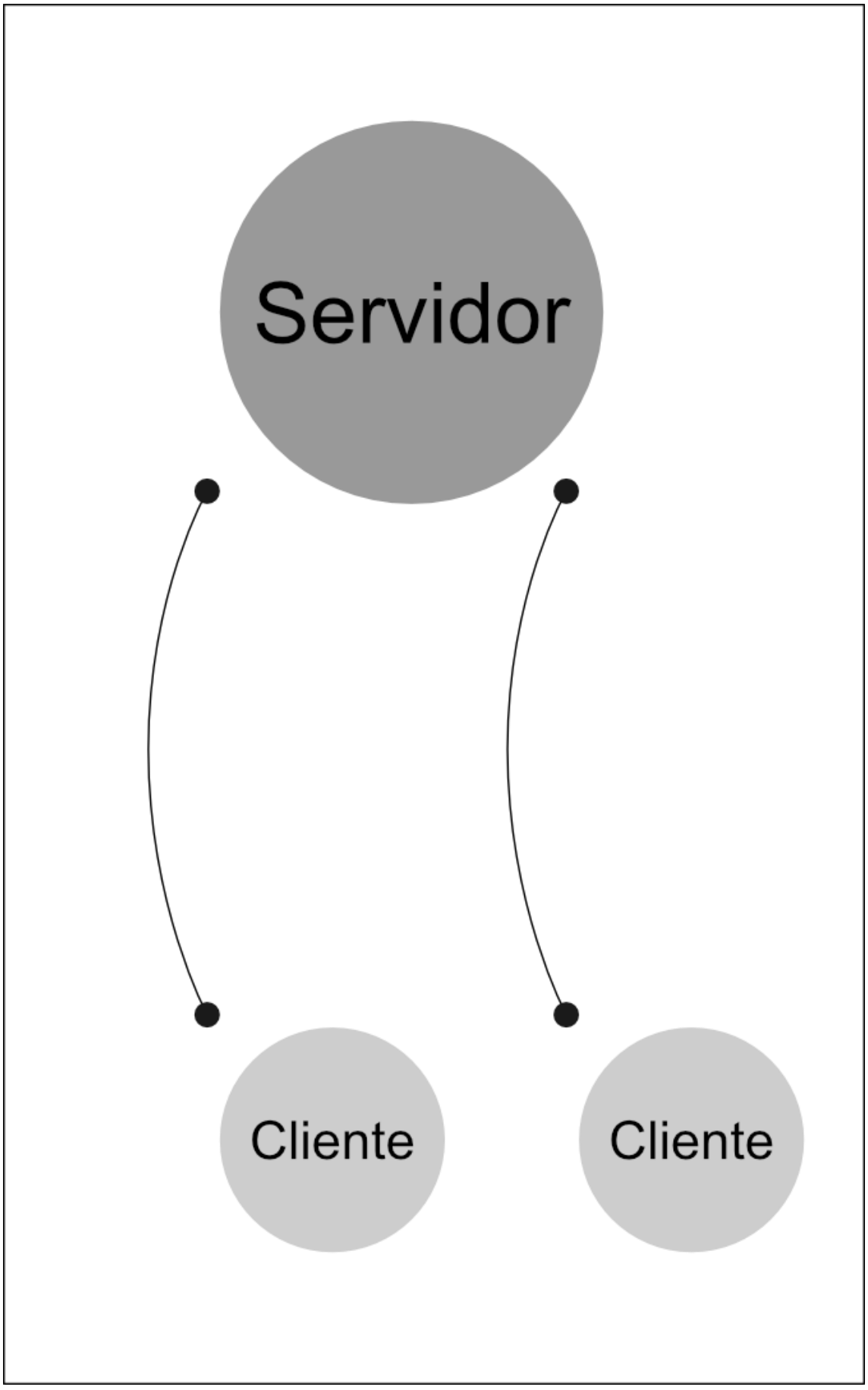

Figura 9 - Diagrama da Arquitectura

9. Conclusões

Com este trabalho concluído, foi possível desenvolver conhecimento sobre os protocolos, tanto de comunicação como para o correcto funcionamento da aplicação. Temos de ter em conta das várias opções possíveis por parte dos Clientes e também ter a segurança e a garantia que o Cliente que estamos a receber não é falsificado.

Reconhecemos a importância do XML e do XSD para a comunicação entre máquinas com uma arquitectura diferente e para salvaguardar qualquer caso impróprio de acção por parte dos Clientes. Mais especificamente, mensagens mal-formadas.

Foi também possível retirar a base com que os serviços e jogos *online* funcionam e como decidem o que deve ser implementado no Cliente e no Servidor. Desta forma, podemos concluir que um Cliente *Thin* é apenas responsável pela apresentação da informação, removendo um enorme peso computacional da máquina em questão.

Desenvolvemos o nosso conhecimento quanto aos *Sockets* e como identificamos as máquinas, a partir do IP. Percebemos que, neste caso, é preferível ter uma ligação persistente ao invés de estabelecer diversas vezes ligações diferentes. Até porque seria mais trabalhoso identificarmos constantemente, dado que as ligações não guardam memória, entre que Cliente esta era estabelecida.

10. Notas Finais

Embora o trabalho esteja desenvolvido e complexo, houve partes em falta, mais concretamente na apresentação do conteúdo na parte do Cliente. Como o trabalho se encontra, assim que um jogo é terminado, não é apresentado esse estado e apenas é revelado a peça vitoriosa quando outra peça é jogada, atualizando o tabuleiro com duas peças de uma vez. Isto também impede o jogador de voltar à lista de Clientes, assim como atualizar a sua informação.

Também ficou em falta a personalização do perfil do Cliente, concretamente, a nacionalidade, a idade e a fotografia. Como mencionado, também, o progresso do Cliente não é guardado.

Todas estas partes serão implementadas para a segunda parte do trabalho, dado que esse é dependente do bom funcionamento desta primeira parte.

Também pretendemos optimizar algumas coisas, por exemplo, alguns métodos da classe Mensagem. Uma classe de utilidade que, por vezes, se torna difícil de perceber o fluxo de operações, assim como os parâmetros a receber e o que retorna. Funcional mas possível de melhorar.

11. Bibliografia

[1] ICD, Slides, “Comunicação entre Processos - Sockets”.

[2] “TCP”, [Online]. “https://pt.wikipedia.org/wiki/TCP/IP”.

[3] “UDP”, [Online]. “https://pt.wikipedia.org/wiki/Protocolo_de_datagrama_do_usu%C3%A1rio”.

[4] ICD, Slides, “Linguagem XSD (XML Schema Definition)”.

[5] “XML”, [Online]. “https://pt.wikipedia.org/wiki/XML”.

[6] ICD, Slides, “Linguagem XSD (XML Schema Definition)”.

[7] “XSD”, [Online]. “https://pt.wikipedia.org/wiki/XML_Schema”.